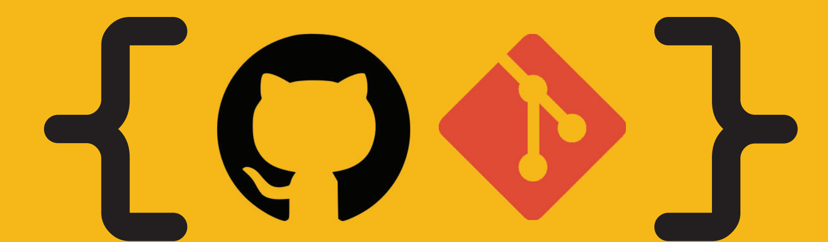


GitHub et Git

UN GUIDE>

Partie d'

⚡ Apprends et Applique


Un projet éducatif produit par
La Fondation Dridi
The Dridi Foundation

Une éducation de qualité, accessible et gratuite facilite la mobilité économique et les changements positifs dans notre monde

Elle offre des opportunités et transforme les vies pour le meilleur

Identifiants

Auteurs et développeurs des cours:
Ing. Ridha Dridi
Mme. Fathia Jelassi Ep Dridi
Aziz Dridi
Salim Dridi

Titre:
GitHub et Git: Un Guide / The Dridi Foundation I La Fondation Dridi

Série:
Série Apprends et Applique
Comprends des références

Mars 2024: Première Édition

À Propos Apprends et Applique, un Projet Éducatif produit par La Fondation Dridi

Apprends et Applique est une plateforme éducative multilingue, concise, accessible et gratuite, qui propose des cours vidéo, des livres et des livres audios sur différents sujets. C'est la première plateforme d'éducation à utiliser l'Intelligence Artificielle pour traduire les livres et produire l'audio des cours. Les cours illustrent le contenu des livres.

Différentes séries sur la Programmation, la Finance et la Comptabilité, les Mathématiques et les Statistiques, les Sciences, les Humanités et les Langues sont offertes.

Pour les premières éditions des livres et des cours vidéo, le contenu est disponible en 11 langues. Les cours vidéo, les livres et les livres audio portant le symbole **β** ont été traduits et leur audio a été généré à l'aide des outils basés sur L'Intelligence Artificielle.

Les versions Française et Arabe ont été produits par l'Ing. Ridha Dridi, Aziz Dridi et Salim Dridi.

Apprends et Applique est également disponible sous forme d'application sur Android et iPhone. Cherchez Learn and Apply dans les App Stores.

fr.LearnApply.org

À Propos La Fondation

L'impact et les productions de la fondation sont un effort collectif de la part des membres de la famille de l'Ing. Ridha Dridi. Mme. Fathia Jelassi Ep Dridi est son épouse et Aziz Dridi et Salim Dridi sont ses fils.

Les projets de la fondation sont dédiés au plus grand bien économique et social de l'humanité **fr.Dridi.org**

Les Autres Projets de La Fondation en 2024

Third Pillar est un site web et une application mobile sur Android et iPhone pour aider les musulmans à calculer et à payer leur aumône obligatoire, la

zakat. Disponible en 11 langues. La Zakat est le troisième des cinq piliers de l'Islam.

fr.ThirdPillar.net

Les familles Dridi et Jelassi viens de la Tunisie. Afin de contribuer à l'amélioration des conditions économiques de la Tunisie et pour offrir des opportunités, la fondation a soumis des demandes de financement à l'USAID, USADF et à d'autres organisations publiques et privées pour financer la production de l'énergie renouvelable. Il a été proposé d'augmenter la production d'énergie solaire photovoltaïque en Tunisie.

Pour voir l'état d'avancement de cette proposition, veuillez visitez

fr.Dridi.org/solaire

L’Engagement de La Fondation envers L’Égalité, la Diversité et L’Inclusion

La Fondation Dridi reconnaît la diversité des lecteurs dans leurs différentes identités, origines et opinions, et le contenu est créé sans parti pris pour ces aspects.

La fondation se consacre à la création de contenus éducatifs dans différentes langues.

Les différences d'apprentissage visuel ont été prises en compte lors de la création des matériaux et des versions accessibles dédiées des livres sont disponibles.

Pour toute inquiétude concernant la partialité ou pour améliorer les aspects liés à l'accessibilité, veuillez contactez

contact@learnapply.org

Informations Légales

Learn and Apply I Apprends et Applique est un projet éducatif produit par La Fondation Dridi I The Dridi Foundation.

The Dridi Foundation
510 Clinton Square
Rochester, NY 14604
United States

Directeurs et propriétaires:
Ridha Dridi, Fathia Jelassi Ep Dridi, Aziz Dridi, Salim Dridi

E-Mail: contact@dridi.org
Téléphone: +1 607-216-7767
Fax: +1 607-360-6587

Les dictionnaires en ligne Cambridge et Collins et les outils basés sur l'intelligence artificielle Google Gemini, Microsoft Copilot, GPT, DeepL, Google Translate, et Microsoft Azure Speech Studio ont contribué à l'élaboration des idées, à la rédaction, à la traduction et à la production audio.

Table des Matières

Chapitre 1
Les Fondamentaux

Contenu

Après une introduction courte au livre et son cours vidéo, le chapitre des fondamentaux commence par l'installation et la configuration des outils et des comptes nécessaires, puis commence avec les bases de Git, y compris ses trois étapes: le répertoire de travail (working directory), la zone de préparation (staging area) et le dépôt (repository), ainsi que ses composants: les branches, les commits, les étiquettes (tags), les arbres (trees), les blobs et les références. Il fournit également un aperçu des commandes **`git`** les plus populaires et conclut avec des techniques avancées telles que combiner plusieurs commits et sauvegarder temporairement les modifications à l'aide de stashing et les arbres de travail (worktrees).

Unité 1.1 Introduction

fr.learnapply.org/course/intro

1.1.1 Introduction au livre

GitHub et Git: Un Guide est un livre de niveau intermédiaire qui couvre le contenu d'un point de vue appliqué. Une version audio de ce livre a été produite.

Ce livre suit le cours vidéo d’Apprends et Applique | Learn and Apply, qui présente le contenu. Le cours est disponible sur différentes plateformes, notamment sur le site web d’Apprends et Applique et sur YouTube.

Learn and Apply est également disponible sous forme d'application sur Android et iOS.

Si vous êtes familier avec GitHub et Git, nous vous recommandons de regarder l'exemple de projet:

fr.learnapply.org/git/demo

Les fichiers de code sont disponibles sur GitHub:

github.com/learnapply/git

Sous chaque unité se trouve un lien vers la page web correspondante d’Apprends et Applique, qui inclut la démonstration vidéo.

La série `Programmation` utilise Windows comme système d’exploitation ; cependant, tout peut être reproduit sur macOS ou Linux.

Ce livre explique la plupart des sujets liés à GitHub et au système de contrôle de version Git. Il décrit de nombreuses commandes Git et détaille les cas d'utilisation courants et les options.

Une expérience précédente n'est pas nécessaire.

Les avantages de l'utilisation de Git pour la gestion des fichiers de code, tels que le branchement et la gestion de l'historique, sont expliqués.

GitHub est abordé à partir de la section 3. Il permet de créer des dépôts à distance et offre des fonctions de collaboration ainsi que des services additionnels tels qu'Actions et Pages.

Dans les projets et les équipes de travail, Git et GitHub ne sont pas utilisés de manière directe, et la plupart des commandes et techniques sont combinées et personnalisées.

C'est en forgeant qu'on devient forgeron. Appliquez le contenu de ce livre à un projet personnel et essayez d'utiliser GitHub et Git dans tous vos futurs développements de logiciels, quel que soit le langage de programmation.

1.1.2 Introduction à GitHub et Git

Git, abréviation de Global Information Tracker, est le système de contrôle de version le plus populaire et possède la plus grande communauté de soutien en ligne pour les développeurs. Ce système est utile pour résoudre les erreurs et mettre en œuvre de nouvelles idées.

En utilisant Git pour un projet, un programmeur ou une équipe de programmeurs peut suivre, comparer et annuler les modifications apportées aux fichiers.

Git comprend des fonctions qui facilitent l'intégration des modifications apportées par différents développeurs dans la base de code principale d'un projet.

Il contribue également à l'audit et à la conformité grâce à ses fonctions de gestion de l'historique. Il enregistre les personnes qui ont apporté des modifications, le moment où elles l'ont fait et la justification fournie par les programmeurs sous la forme de messages de validation. Les messages de validation sont similaires aux commentaires de code utilisés dans les langages de programmation.

Git peut être utilisé sur des ordinateurs personnels en utilisant exclusivement des répertoires locaux. Cependant, la plupart des programmeurs et des équipes préfèrent l'utiliser avec une solution en nuage, comme GitHub.

GitHub propose des serveurs gratuits et ajoute des fonctionnalités de répertoire à distance. En outre, il offre des fonctionnalités complémentaires à Git, telles que les demandes d'extraction (Pull Requests), à ne pas

confondre avec la commande **`git pull`**, et d'autres fonctions de collaboration et de partage si l'on travaille en équipe.

Git est le système de contrôle de version le plus populaire et GitHub est la solution cloud la plus populaire pour les projets basés sur Git.

Unité 1.2
Installation et configuration



fr.learnapply.org/git/1.2

1.2.1 Git Bash

Le programme Git Bash regroupe tous les outils nécessaires pour utiliser Git et GitHub à partir de la ligne de commande.

Windows **gitforwindows.org**

macOS, Linux **git-scm.com**

Une fois installées, les commandes Git peuvent être exécutées soit dans l'interface utilisateur en ligne de commande Git Bash, soit dans la ligne de commande des systèmes d'exploitation Windows, macOS ou Linux.

Pour Mac, Homebrew, un gestionnaire de paquets pour la gestion des logiciels, peut également être utilisé pour installer Git.

macOS **brew.sh**

Une fois Homebrew installé, tapez **`brew install git`** dans la ligne de commande pour installer Git.

1.2.2 Configurations mondiales

`git config` permet de configurer certains paramètres lors de l'utilisation de Git. L'option **`--global`** permet de les configurer globalement. Les configurations globales sont placées dans le fichier .gitconfig.

Code 1 Configurations globales communes

Exemples	Descriptions

`git config --global core.editor "code --wait"`	Définit Visual Studio Code comme le programme par défaut pour manipuler les fichiers avec les commandes Git.
`git config --global user.name "nom d'utilisateur"`	Configure le nom d'utilisateur Git
`git config --global user.email "email"`	Configure l'email Git
`git config --global --add -bool push.autoSetupRemote true`	Crée une branche distante lors du transfert des modifications si la branche distante nommée n'existe pas.
`git config --global help.autocorrect 20`	Autocorrection des erreurs de commande Git après deux secondes
`git config --global rerere.enabled true`	Activation de rerere
`git config --list`	Vérifie que tout a été configuré correctement

Réutiliser la résolution enregistrée (rerere) demande à Git de se souvenir de la façon dont il a résolu un conflit 'hunk' afin que la prochaine fois qu'il rencontre le même problème, Git puisse le résoudre automatiquement.

Un conflit hunk est une situation dans laquelle deux versions différentes d'un fichier sont créées, les programmeurs les modifient différemment et Git ne peut pas déterminer automatiquement comment fusionner les modifications.

1.2.3 GitHub

Pour utiliser GitHub, le programmeur doit créer un compte sur

github.com

Une fois le compte créé, les développeurs doivent créer un jeton d'accès personnel (PAT) dans les paramètres de leur compte GitHub.

GitHub a mis fin à l'utilisation des mots de passe de compte comme méthode d'authentification dans la ligne de commande en 2021. Les PAT sont désormais utilisés à la place.

1.2.4 Éditeur de code et extensions

Visual Studio Code, communément appelé VS Code, est le programme d'édition de code le plus populaire. Il prend en charge la plupart des langages de programmation et offre une intégration avec d'autres outils de programmation largement utilisés.

Windows, macOS, Linux **code.visualstudio.com**

Des fonctionnalités supplémentaires peuvent être ajoutées à VS Code à l'aide d'extensions. Dans VS Code, elles sont installées à partir de l'onglet "extensions" dans la barre verticale de gauche.

Ils peuvent également être installés à partir de la place de marché

marketplace.visualstudio.com

Voici les extensions VS Code utilisées dans le cours vidéo GitHub et Git d’Apprends et Applique.

- **Prettier**: Formate la syntaxe des codes de différents langages de programmation et les rend plus faciles à lire.
- **GitHub Repositories**: Permets de gérer (parcourir, rechercher, modifier et livrer) des dépôts distants hébergés sur GitHub.
- **GitHub Pull Requests and Issues** : Permets la gestion et la révision des Pull Requests de GitHub.
- **GitHub Actions**: Aide à gérer les flux de travail des actions GitHub

Les autres modules complémentaires les plus populaires sont GitDoc et Live Share. GitDoc permet de valider, de pousser et d'extraire automatiquement les modifications lorsque les fichiers sont enregistrés. Live Share permet une collaboration en temps réel lorsque l'on travaille en équipe.

Unité 1.3
Les 3 étapes de Git



fr.learnapply.org/git/1.3

Figure 1 Vue d'ensemble des trois étapes par lesquelles passent les fichiers dans les projets Git



1.3.1 Working Directory - Répertoire de travail

Le répertoire de travail est l'endroit où tous les fichiers sont stockés et où les modifications sont effectuées en premier.

Ces fichiers et dossiers peuvent se trouver dans trois états :

- **Suivi** : Ils ont été précédemment déposés dans un dépôt.
- **Non suivi** : Ils n'ont pas été précédemment déposés dans un dépôt.
- **Ignoré** : Le programmeur a désactivé le suivi de Git pour eux. La façon de procéder est décrite à la page 50.

1.3.2 Staging Area - Zone de transit

La zone de transit, parfois appelée index, est une zone intermédiaire entre le répertoire de travail et le dépôt. Les modifications doivent d'abord être ajoutées à la zone de transit à l'aide de `git add` avant d'être livrées au dépôt.

La zone de transit permet aux programmeurs de regrouper de nombreuses modifications dans une livraison ciblée

.

1.3.3 Repository - Dépôt

Un dépôt, souvent appelé "repo", est une structure de données gérée par Git. Il s'agit du dossier contenant les fichiers et les sous-dossiers du projet.

Les dépôts peuvent être locaux ou distants si le programmeur utilise GitHub.

Pour sauvegarder les modifications qu'il a effectuées, le programmeur doit les valider à l'aide de **`git commit`**. Le dépôt est l'endroit où les modifications sont sauvegardées après avoir été validées à partir de la zone de mise à disposition.

Le dépôt suit les modifications et contient tous les objets des commits et leurs références. Il stocke également les métadonnées des fichiers et des dossiers du projet Git.

Unité 1.4
Les composants de Git



fr.learnapply.org/git/1.4

1.4.1 Branches

La création de branches équivaut à la création de versions d'une base de code. En utilisant des branches, les programmeurs peuvent mettre en œuvre de nouvelles fonctionnalités et corriger des erreurs risquées en isolation de la branche de production.

Pour intégrer les modifications de code effectuées dans d'autres branches à la branche principale, les commandes **git merge**, **git rebase** ou **`git cherry-pick`** sont utilisées.

Pour référencer une branche dans différentes commandes Git, utilisez son nom ou les premiers caractères du hash du commit effectué sur cette branche. Le hash est obtenu à l'aide de la commande

```
git log
```

Outre le hachage, **`git log`** affiche d'autres informations, notamment le nom de la branche sur laquelle on travaille actuellement, ainsi que la date et l'heure auxquelles le programmeur a effectué le commit.

Une alternative à **`git log`** est **`git reflog`**, discuté à la page 27.

En plus de **`git branch`**, il existe d'autres commandes pour différentes fonctions de gestion des branches.

Code 2 Cas d'utilisation courants de la gestion des succursales

Exemples	Descriptions
`git branch nom_de_la_branche`	Création d'une nouvelle branche
`git branch --contains commit_hash`	Liste les branches contenant un certain commit
`git branch -vv`	Voir une branche plus en détail
`branche git`	Liste des agences locales
`git branch -d nom_de_la_branche`	Supprime la branche
`git branch -all`	Liste des branches locales et distantes
`git branch -sort=committerdate`	Affiche une liste de toutes les succursales locales et les trie en fonction de la date de leur dernière validation.
`git branch nom_de_la_première_branche..nom_de_la_deuxième_branche`	Affiche toutes les différences faites sur la deuxième branche depuis qu'elle a divergé de la première branche
`git checkout nom_de_la_branche`	Passage à la branche nommée
`git clean -fd`	Rejette tous les changements non suivis de la branche courante
`git rev-parse -abbrev-ref HEAD`	Imprime le nom de la branche actuelle
`git difftool -t meld --dir-diff`	Affiche les changements effectués dans le répertoire de travail
`git difftool -t meld --dir-diff first_commit_hash second_commit_hash`	Affiche les différences entre deux commits
`git branch -a -merged`	Affiche une liste de toutes les succursales locales

Une bonne pratique consiste à définir des branches différentes pour des tâches différentes et à ne pas avoir trop de branches. Cela permet aux équipes de travailler sur une fonctionnalité de différentes manières si elles le souhaitent.

1.4.2 Engagements

Git possède 4 types d'objets : les commits, les tags, les arbres et les blobs.

Pour sauvegarder les modifications apportées, le programmeur doit enregistrer ses modifications en les validant à l'aide de la commande

git commit -m "commit_message"

Avant la validation, l'ajout de fichiers à la zone d'essai à l'aide de la fonction

git add

Une fois livrés, ils sont suivis par le système Git.

En plus du message de validation, chaque validation contient :

- hachages de l'arbre et du commit parent
- le nom et l'adresse électronique du programmeur

Les hachages de l'arbre et du commit parent sont des métadonnées et des pointeurs vers d'autres objets Git qui contiennent le contenu réel du commit.

Comme pour les branches, les commits sont référencés dans d'autres commandes git en utilisant les premiers caractères du hash du commit. Le hash est trouvé en utilisant l'une ou l'autre des commandes suivantes

```
git log
git reflog
```

Bien qu'il soit possible d'utiliser n'importe quel nombre de caractères, il est préférable d'en utiliser au moins 7 afin d'éviter que Git ne confonde différents commits dans les grands projets.

Code 3 Cas d'utilisation courants de **git commit** et commits

Exemples	Descriptions

`git cat-file -p commit_hash`	Consulter le contenu d'un commit
`git commit -m commit_message`	Crée un nouveau commit
`git commit rev-list --online branch1 branch2 ^branch3`	Liste les commits dans branch1 et branch2, mais pas dans branch3
`git show`	Elle montre le commit précédent
`git show @~n`	Affiche le nième commit à partir du dernier commit effectué
1. `git checkout main` 2. `git add .` 3. `git commit -m "commit_message"` 4. `git branch nom_de_la_branche` 5. `git reset HEAD~n --hard` 6. `git checkout nom_de_la_branche`	Un exemple de déplacement de commits de la branche principale vers une nouvelle branche. Cela n'affecte pas les changements déjà poussés vers un dépôt GitHub.

Pour améliorer la productivité, les livraisons doivent être les plus petites possibles et chaque livraison peut être résumée en une simple phrase.

1.4.3 Messages d'engagement

Le message de validation est un commentaire laissé par un programmeur pour expliquer les modifications apportées, afin que d'autres programmeurs de l'équipe puissent s'y référer ultérieurement. Ils sont similaires aux commentaires de code dans les langages de programmation.

Pour rédiger de bons messages d'engagement, suivez les règles suivantes

1. Utiliser le présent et l'impératif
2. Être cohérent, clair et descriptif
3. Le message doit être un résumé en ligne

4. Fournir un contexte pour les changements apportés au cours de la durée de vie du projet
5. Respecter la grammaire et la ponctuation

Voici quelques exemples de l'humeur impérative :

- "Corriger les bogues" au lieu de "Corriger les bogues"
- "Mise à jour du document" au lieu de "Mise à jour du document".

Exemples de respect de ces règles :

- Faire x faire z
- Fusionner la Pull Request #123 dans a de b
- Si j'applique ce commit, il fera z
- Ajouter la fonction b

Pour éviter de retaper les mêmes informations, les programmeurs peuvent configurer un modèle par défaut, sous la forme d'un fichier texte, pour tous les futurs messages de validation à l'aide de la commande

git config commit.template nom_du_fichier

1.4.4 Tags

Les balises sont des références qui renvoient à des points antérieurs dans l'histoire d'un projet Git. Ils ressemblent aux numéros de version des programmes.

Code 4 Cas d'utilisation courants des balises

Exemples	Descriptions
`git tag nom_du_tag`	Crée un tag de la branche courante
`git tag tag_name commit_hash`	Crée un nouveau commit
`git tag`	Liste de tous les tags créés
`git describe`	Recherche le tag le plus récent d'un commit

1.4.5 Arbres

Les arbres représentent la structure d'un dossier et son contenu.

Bien qu'ils soient un élément essentiel dans la façon dont Git stocke et gère les versions de code, aucun cas d'utilisation courant ne peut être accompli

avec eux.

1.4.6 Blobs

Les blobs contiennent les données brutes de chaque fichier du dépôt. Il n'y a pas de cas d'utilisation courante les concernant.

1.4.7 Le dossier .git

Il s'agit d'un dossier caché dans le répertoire principal de chaque projet Git. Il est créé immédiatement après l'initialisation. L'initialisation se fait à l'aide de **git init**

Les informations d'un dépôt sont stockées dans 3 sous-dossiers du dossier .git :

- Objets
- Références
- Têtes

Ce dossier contient toutes les informations et métadonnées nécessaires à la gestion de l'historique des versions d'un projet. Il comprend les composants de métadonnées suivants, essentiels pour les opérations de contrôle de version d'un projet

Configurations	**Le pointeur de référence HEAD**	**Crochets**
Journaux	Blobs	Tags
Branches	Engagements	Arbres

Pour afficher le contenu du dossier .git, utilisez la commande

ls -a .git

ou **ls -la .git** pour plus de détails

Dans ces commandes, l'option **-a** permet d'afficher les fichiers et dossiers cachés dans la ligne de commande.

Pour voir ce dossier dans l'explorateur de fichiers de votre système d'exploitation, activez la possibilité d'afficher les fichiers et dossiers cachés.

1.4.8 Le pointeur HEAD

Les références sont la méthode par laquelle Git pointe vers une branche. La référence HEAD pointe vers la branche sur laquelle on travaille actuellement.

Pour afficher le contenu du fichier HEAD, utilisez la commande

cat .git/HEAD

Parfois, au lieu d'une seule ligne contenant la branche en cours, la sortie de cette commande sera constituée de texte et de nombres aléatoires ; dans ce cas, le pointeur HEAD est détaché.

Il s'agit d'une situation dans laquelle la référence ne pointe vers aucune branche et qui se produit généralement lorsqu'un programmeur crée une nouvelle livraison sans créer de nouvelle branche.

Pour y remédier, on utilise les commandes **git checkout** ou **git switch.**

Unité 1.5 Commandes Git couramment utilisées et leurs options



fr.learnapply.org/git/1.5

Pour obtenir plus de détails et d'informations sur ces commandes, faites-les suivre de `-h` ou utilisez la commande `git help`. Par exemple, pour plus d'informations sur `git rm`, utilisez les commandes

```
git rm -h
git help rm
```

Consultez toujours la documentation officielle de Git pour obtenir une liste complète des options de commande, des explications détaillées et les dernières modifications.

git-scm.com/docs

1.5.1 `git init`

`git init` est la première commande tapée pour démarrer un projet Git dans le dossier courant de la ligne de commande. En option, elle peut prendre le nom d'un dossier en argument.

1.5.2 `git status`

`git status` affiche l'état actuel du répertoire de travail et de la zone de transit. En option, ajoutez **`-sb`** comme argument pour afficher la forme courte de la même sortie.

1.5.3 `git add`

`git add` ajoute les modifications effectuées dans le répertoire de travail à la zone de stockage.

Code 5 Cas d'utilisation courants de **`git add`**

Exemples	Descriptions
`git add .`	Ajoute tous les fichiers à la zone de transit
`git add -u`	Ajoute les modifications des fichiers déjà présents dans la zone de transit sans les fichiers nouvellement créés dans le répertoire de travail.
`git add file1 file3`	Ajoute les fichiers sélectionnés à la zone de transit
`git add -i`	Une vue interactive est affichée à l'invite de commande et le programmeur peut sélectionner les fichiers à ajouter

Le dernier cas d'utilisation est la version interactive de l'ajout. Elle doit être utilisée lorsqu'il s'agit de modifications enchevêtrées dans le répertoire de travail qu'un programmeur souhaite répartir dans différentes livraisons.

Il devrait également être utilisé si le programmeur est au milieu d'un rebasement interactif et souhaite diviser un commit trop important.

1.5.4 `git commit`

`git commit` enregistre les modifications effectuées dans le dépôt local. Les modifications effectuées dans le répertoire de travail doivent d'abord être ajoutées à la zone de mise à disposition avant d'être enregistrées dans le dépôt. Le programmeur doit spécifier un message de validation.

```
git commit -m "commit_message"
```

1.5.5 `git branch`

`git branch` crée des branches. Créer des branches équivaut à créer des versions d'une base de code. En utilisant des branches, les programmeurs peuvent implémenter de nouvelles fonctionnalités et corriger des bogues à risque en s'isolant de la branche de production.

Code 6 Cas d'utilisation courants de **`git branch`**

Exemples	Descriptions
`git branch nom_de_la_branche`	Création d'une nouvelle branche
`git branch --contains commit_hash`	Liste les branches contenant un certain commit
`git branch -vv`	Voir une branche plus en détail
`branche git`	Liste des agences locales
`git branch -d nom_de_la_branche`	Supprime la branche
`git branch -all`	Liste des branches locales et distantes
`git branch -sort=committerdate`	Affiche une liste de toutes les succursales locales et les trie en fonction de la date de leur dernière validation.
`git branch nom_de_la_premièr`	Affiche toutes les différences faites sur la deuxième branche

`e_branche..nom_de_la_deuxième_branche`	depuis qu'elle a divergé de la première branche

1.5.6 `git mv`

`git mv` renomme les dossiers, les dépôts et les branches locales et distantes. Il peut également être utilisé pour déplacer des fichiers et des dossiers entre différents emplacements.

Par exemple, pour modifier l'emplacement d'un fichier et l'ajouter à la zone de transit, utilisez la commande suivante

```
git mv chemin_existant chemin_nouveau chemin
```

1.5.7 `git rm`

git rm supprime un fichier du projet et ajoute son retrait à la zone de préparation de la prochaine livraison.

```
git rm nom_du_fichier
```

1.5.8 `git log`

`git log` est utilisé pour afficher les détails des livraisons. Il affiche le hash du commit, la branche sur laquelle on travaille actuellement, la date et l'heure du commit, ainsi que le nom et l'email du programmeur qui a effectué le commit.

Code 7 Cas d'utilisation courants de **`git log`**

Exemples	Descriptions
`git log`	Affiche une liste de toutes les modifications dans l'ordre chronologique.
`git log --stat -M`	Affiche tous les journaux de validation avec une indication des chemins qui ont été déplacés.

`git log -follow nom_du_fichier`	Liste l'historique des versions d'un fichier, y compris l'historique des renommages.

1.5.9 `git tag` et `git describe`

Les commandes **`git tag`** et **`git describe`** permets de travailler avec les balises. Les balises sont des références qui renvoient à des points antérieurs dans l'historique d'un projet Git. Elles ressemblent aux numéros de version des programmes.

Code 8 Cas d'utilisation courants des balises

Exemples	Descriptions
`git tag nom_du_tag`	Crée une étiquette de la branche actuelle
`git tag tag_name commit_hash`	Crée un nouveau commit
`git tag`	Liste de tous les tags créés
`git describe`	Recherche l'étiquette la plus récente d'un commit

1.5.10 `git diff`

git diff est utilisé pour visualiser les différences entre les fichiers et les branches.

Code 9 Cas d'utilisation courants de **`git diff`**

Exemples	Descriptions
`git diff`	Affiche les différences entre les versions stagées et non stagées des fichiers.
`git diff -staged`	Affiche les différences entre les fichiers de la zone de transit et la

	dernière version du répertoire de travail.
`git diff première_branche deuxième_branche`	Montre les différences entre deux branches

1.5.11 `git blame`

`git blame` est utilisé pour annoter chaque ligne d'un fichier avec la date de création de la ligne et l'auteur de la modification.

```
git blame nom_du_fichier
```

1.5.12 `git show`

`git show` est utilisé pour afficher les détails des objets Git (blobs, arbres, tags et commits).

TableAU 1 Résultats de **`git show`** lorsqu'il est utilisé avec différents objets Git

Objets	Sorties
Commit - Engagement	Affiche le message du journal et les autres changements survenus lors de la validation sélectionnée.
Tag - Étiquette	Affiche le message de la balise et les autres balises incluses dans la balise
Tree - Arbre	Afficher les noms et le contenu des objets dans un arbre
Blob	Affiche le contenu direct du blob

Unité 1.6
Combiner les commits et sauvegarder temporairement les changements



fr.learnapply.org/git/1.6

1.6.1 Écraser

Il s'agit d'une méthode permettant de combiner 2 ou plusieurs commits en un seul, ce qui permet d'obtenir un historique des commits plus propre et de faciliter la révision des modifications. L'écrasement n'est pas une opération dédiée à Git, mais est plutôt utilisé lors de la fusion de modifications provenant d'autres branches dans la branche principale d'un projet.

Elle peut être utilisée comme option avec les commandes **`git merge`** et **`git rebase`**. Elle peut également être utilisée pour écraser des commits récents.

Code 10 Cas d'utilisation courants de l'écrasement

Exemples	Descriptions
1. `git reset --soft HEAD~n` 2. `git commit`	Squash n commits récents sans rebasage
1. `git merge --squash nom_de_la_branche` 2. `git commit`	Squash pendant une fusion. Comme alternative à la ligne de commande, GitHub fournit un moyen d'écraser pendant une fusion
`git rebase -i commit_hash`	Squash lors d'un rebasement interactif. L'éditeur interactif permettra au programmeur de choisir le commit à écraser et l'invitera à saisir un message de commit. De nombreux programmeurs choisissent d'utiliser le message de validation pour indiquer les

	validations qui ont été écrasées et l'endroit à partir duquel la refonte doit commencer

1.6.2 Mise à l'écart

La mise en réserve est une méthode permettant de stocker temporairement les modifications apportées au répertoire de travail afin que le programmeur puisse travailler sur autre chose et reprendre le travail sur les modifications plus tard.

Code 11 Cas d'utilisation courants de la mise en réserve

Exemples	Descriptions
`git stash`	Crée une nouvelle cache et enregistre toutes les modifications non validées dans le répertoire de travail et la zone de mise à disposition.
`git stash push -u`	Similaire à **`git stash`**, mais inclut également les fichiers non suivis. Les fichiers non suivis sont ceux qui n'ont jamais été dans un dépôt.
`git stash branch branch_name`	Crée une nouvelle réserve et crée une nouvelle branche à partir de celle-ci avec le nom spécifié
`git stash list`	Liste de toutes les réserves faites
`git stash apply`	Applique les modifications de la réserve la plus récente au répertoire de travail, mais conserve la réserve intacte.

1.6.3 Worktrees

Les Worktrees sont des dossiers dans un projet Git où les programmeurs stockent temporairement les modifications. C'est une alternative à **`git stash`** qui a la même fonction.

Code 12 Cas d'utilisation courants de **git worktree**

Exemples	Descriptions
1. `git worktree add -b emergency_fix ../temp master` 2. `pushd ../temp` 3. `git commit -a -m commit_message` 4. `popd` 5. `rm -rf ../temp` 6. `git worktree prune`	Effectuez une correction d'urgence, retirez-la si nécessaire, puis reprenez la session de codage précédente.
`git worktree prune`	Supprime les informations relatives à un arbre de travail
`git worktree list`	Liste des arbres de travail existants

Chapitre 2 Gestion de l'historique et annulation des modifications

Contenu

Ce chapitre aborde différentes fonctionnalités de Git, notamment les opérations d'annulation, de récupération, de modification et de restauration. Il aborde les commandes de gestion des branches telles que **`checkout`**, **`switch`** et **`restore`**, et présente une autre façon de visualiser l'historique des livraisons en utilisant git reflog. Le chapitre se termine par une explication des correctifs, qui capturent et permettent de partager, d'appliquer ou de réviser les modifications apportées à une base de code.

Unité 2.1
Défaire et changer



fr.learnapply.org/git/2.1

2.1.1 Annuler un commit

Si des erreurs ont été commises dans les modifications apportées à un dépôt local ou distant, il est possible de revenir en arrière en utilisant **`git revert`**. Il annule une livraison et préserve l'historique des versions.

`git revert` crée un nouveau commit basé sur l'inverse des changements du commit avec les erreurs.

Code 13 Cas d'utilisation courants de **`git revert`**

Exemples	Descriptions
`git revert HEAD`	Annule les modifications de la dernière livraison
`git revert commit_hash`	Annule une certaine livraison

2.1.2 Modifier les commits

La commande **`git commit --amend`** modifie le commit précédent en y ajoutant les modifications de la zone de transit.

Voici un exemple.

Code 14 Le programmeur modifie deux fichiers et oublie d'en valider un.

```
1. git add file1
2. git commit
```

le programmeur se rend compte qu'il a oublié de valider le deuxième fichier

```
3. git add file2
4. git commit --amend
```

En tant que meilleure pratique dans le cadre d'un travail d'équipe, ne modifiez pas les commits sur lesquels les autres programmeurs ont basé leur travail afin d'éviter les conflits de fusion.

2.1.3 Annuler les modifications

`git reset` annule les modifications.

Code 15 Cas d'utilisation courants de `git reset`

Exemples	Descriptions
`git reset commit_hash`	Réinitialise la branche en cours au commit spécifié
`git reset --hard commit_hash`	Réinitialise la branche courante au commit spécifié et change le répertoire de travail.
`git reset @~n`	Réinitialise la branche en cours à l'état dans lequel elle se trouvait lors de la dernière livraison.
1. `git reset --soft HEAD~n` 2. `git commit`	Annule toutes les modifications effectuées depuis la dernière livraison. Cela ne supprime pas la livraison de l'historique.
`git reset --hard`	Annule les modifications ajoutées à la zone de transit
`git reset --hard commit_hash`	Supprime les modifications non validées du répertoire de travail et de la zone de transit et réinitialise la branche courante à la validation identifiée. Les livraisons suivantes sont rejetées
`git reset --`	Déstockage des fichiers ajoutés à la zone de transit

2.1.4 Modifier le nom et l'adresse électronique du développeur

Pour ce faire, créez d'abord un fichier filter.sh.

Code 16 Contenu de filter.sh

```
1. if [ "$GIT_AUTHOR_NAME" = "Author to Change From" ]
2. then
3. export GIT_AUTHOR_NAME= "Auteur à modifier en"
4. export GIT_AUTHOR_EMAI L="email.to.change.to@example.com"
5. fi
```

Après avoir modifié le contenu avec le nom et l'adresse électronique, exécutez le fichier

Code 17 Exécution de filter.sh

```
chmod +x ./filter.sh
```

Revenir à un commit précédent

ALGORITHME 1 Étapes pour revenir à une validation précédente

1 : Trouver le hash du commit en utilisant le **git log**
2 : Détacher le pointeur HEAD en utilisant **git checkout** ou **git switch**
3 : [Optionnel] Créer une branche séparée en utilisant **git branch** ou **git checkout -b**

Récupération d'une réinitialisation de git

Dans le cas où le programmeur a exécuté **git reset @~n**, qui réinitialise la branche courante à l'état dans lequel elle se trouvait il y a n commits, a été exécuté et que le programmeur souhaite récupérer, suivez les deux étapes ci-dessous.

ALGORITHME 2 Étapes de récupération après une réinitialisation

1 : Exécutez **git reflog** ou **git log** pour trouver le hash du commit sur lequel revenir.
2 : Utiliser **git reset** commit_hash

Récupération de git stash

Dans le cas où le programmeur a exécuté **git stash,** et qu'il souhaite récupérer la dernière version de stash, utilisez le code **git stash apply.**

Vous pouvez également sélectionner la réserve à partir de laquelle vous souhaitez récupérer les données en suivant ces deux étapes.

ALGORITHME 3 Étapes de récupération d'une cachette

1 : Exécuter **git stash list** pour voir tous les stashs dans une liste
2 : Sélectionner le stash à restaurer en utilisant **git stash apply stash{n}**

Unité 2.2
Comprendre l'extraction, la commutation et la restauration



fr.learnapply.org/git/2.2

2.2.1 git checkout

La commande **git checkout** à plusieurs objectifs.

Bien qu'il soit principalement utilisé pour passer à une branche, il possède d'autres options qui lui permettent de remplir différentes fonctions telles que la suppression de modifications, leur annulation et la restauration de fichiers.

Si le programmeur passe à une branche qui n'existe pas en utilisant **git checkout**, la branche sera créée.

Code 18 Cas d'utilisation courants de **git checkout**

Exemples	Descriptions
`git checkout -b nom de la branche`	Création d'une nouvelle branche et passage à celle-ci

`git checkout -b new_branchname other_branchname`	Création d'une nouvelle branche basée sur une autre branche et passage à la nouvelle branche
`git checkout --file`	Supprime toutes les modifications apportées à un certain fichier
`git checkout stash@{n} -file`	Restauration d'un fichier sélectionné à partir d'une réserve
`git checkout -b nom de la branche HEAD~n`	Crée une nouvelle branche et fixe son commit de base à un commit qui est n ancêtres derrière le commit actuel.
`git checkout -- .`	Annule toutes les modifications effectuées dans le répertoire de travail actuel

2.2.2 Changement de git

`git switch` est une alternative à checkout et peut également être utilisé pour passer à une branche. Cependant, si la branche n'existe pas, elle ne sera créée que si l'option **`-c`** est incluse.

Code 19 Cas d'utilisation courants de **`git switch`**

Exemples	Descriptions
`git switch nom_de_la_branche`	Changement de branche
`git switch -`	Passage à la branche précédente
`git switch -c new_branchname`	Création d'une nouvelle branche et passage à celle-ci

Avec **`git checkout`**, cette commande peut être utilisée pour corriger un pointeur HEAD détaché, discuté à la page 12.

2.2.3 `git restore`

`git restore` restaure un fichier ou un dossier à un état antérieur. Il peut également être utilisé pour dépiler les fichiers ajoutés à la zone de transit et pour supprimer les modifications locales non validées.

Code 20 Cas d'utilisation courants de **`git restore`**

Exemples	Descriptions
`git restore nom_du_fichier`	Rétablis l'état antérieur d'un fichier en supprimant les modifications apportées dans le répertoire de travail.
`git restore nom_du_dossier`	Restaure un dossier dans un état antérieur en supprimant les modifications apportées au répertoire de travail.
`git restore --staged nom_du_fichier`	Déstabilise les modifications ajoutées à la zone de transit et les restaure dans l'état où elles se trouvaient lors de la dernière livraison.
`git restore --source= -c new_branchname`	Création d'une nouvelle branche et passage à celle-ci

Unité 2.3
Afficher l'historique des livraisons en utilisant git reflog



fr.learnapply.org/git/2.3

Reflogs (reference logs) est le nom donné au mécanisme par lequel Git garde la trace des mises à jour de fichiers. Ils indiquent où les références Git ont été mises à jour dans le dépôt local et sont stockés dans le dossier .git.

Après avoir réécrit l'historique à l'aide des techniques décrites à partir de la page 21, un reflog contient des informations sur l'ancien état des branches et permet aux programmeurs de revenir à un état antérieur si nécessaire.

`git log` ne montre pas l'historique des commits qui ne sont pas référencés par une branche ou un tag. **`git reflog`** résout ce problème.

Code 21 Cas d'utilisation courants de **`git reflog`**

Exemples	Descriptions
`git reflog`	Affiche les reflogs effectués dans le dépôt local
`git reflog show nom_branche`	Affiche les entrées de reflog pour la branche
`git reflog stash`	Sauvegarde l'état actuel de l'arbre de travail et l'index de la base de données.
`git reflog expire`	Expire toutes les entrées du journal avant l'expiration par défaut de 90 jours.
`git reflog delete`	Supprime toutes les entrées du reflog
`git reflog --date relative`	Affiche le reflog avec une date relative. Par exemple, "il y a 2 semaines"
`git reflog main@{0}`	Affiche une liste de tous les changements récents apportés au pointeur HEAD
`git reflog show HEAD`	Affiche tous les reflogs de la branche sur laquelle pointe le pointeur HEAD. Il s'agit de la branche sur laquelle le programmeur travaille actuellement.
`git reflog show --all`	Affiche tous les reflogs effectués

Le reflog ne fournit un filet de sécurité que si les modifications ont été validées dans le dépôt local. Tous les reflogs ont une expiration par défaut de 90 jours qui peut être prolongée.

Unité 2.4 Patchs



fr.learnapply.org/git/2.4

Un patch est un fichier texte contenant les différences entre deux fichiers. Il capture les modifications apportées à une base de code et permet au programmeur de les partager, de les appliquer ou de les réviser.

Code 22 Cas d'utilisation courants des correctifs

Exemples	Descriptions
`git am nom_du_patch.patch`	Applique le correctif sous la forme d'un commit
`git am *.patch`	Applique tous les fichiers correctifs à l'arbre
`git format-patch commit_hash`	Convertit tous les commits depuis le commit référencé en patch fichiers
`git apply nom_du_patch.patch`	Applique les modifications du fichier patch au répertoire de travail actuel

Chapitre 3 GitHub et la fusion des modifications

Contenu

Cette section explore GitHub et la fusion des modifications, couvrant les commandes git pour la gestion des dépôts distants, l'intégration des modifications provenant de différentes branches et la résolution des conflits de fusion. Elle présente une alternative à la fusion et au rebasage : la commande `git cherry-pick`. Le chapitre détaille également les Pull Requests, une fonctionnalité clé de GitHub, explique cinq flux de travail de collaboration courants et se termine par une discussion sur la configuration et l'utilisation des connexions SSH.

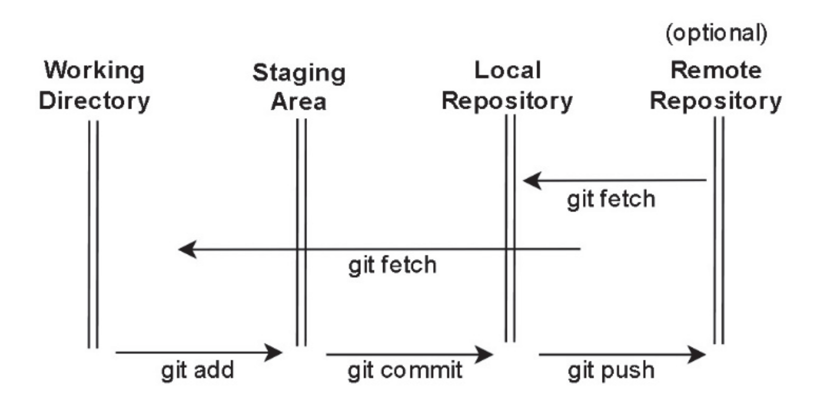
Unité 3.1
Manipuler des dépôts distants et fusionner



fr.learnapply.org/git/3.1

3.1.1 Aperçu

Figure 2 Vue d'ensemble des commandes Git permettant de déplacer des modifications



Les fichiers sont édités par le programmeur dans le répertoire de travail. Pour les préparer à la prochaine validation, ils sont ajoutés à la zone de mise à disposition (Staging Area). Pour enregistrer les modifications, ils sont validés. Les modifications peuvent ensuite être transférées vers un dépôt distant proposé par GitHub.

Afin d'utiliser les dépôts distants, ceux-ci doivent d'abord être liés au dépôt local à l'aide de la commande **git remote add**

3.1.2 git remote

git remote gère les connexions entre les dépôts locaux et distants hébergés sur GitHub. Une fois qu'un dépôt distant a été lié au dépôt local, les modifications peuvent être poussées et récupérées.

Code 23 Cas d'utilisation courants de **git remote**

Exemples	Descriptions
git remote -v	Liste de tous les dépôts distants actuellement configurés avec leur URL
git remote add remote_repo_name url	Lier un dépôt distant au dépôt local
git remote remove remoe_repo_name	Supprime une configuration distante du dépôt local.

3.1.3 clone git

git clone crée une copie locale d'un dépôt distant en utilisant l'URL GitHub du dépôt distant. Cette commande permet à une personne de travailler sur le code de manière indépendante et d'apporter des modifications au dépôt d'origine.

Code 24 Cas d'utilisation courants de **git clone**

Exemples	Descriptions
git clone url	Affiche les différences entre les versions stagées et non stagées des fichiers.
git clone url -b nom_de_la_branche chemin_du_répertoire	Clone un dépôt Git à partir de l'adresse dans le dossier donné et vérifie la branche donnée.
git clone url -b nom_de_la_branche --single-branch	Clone une seule branche

3.1.4 git push

git push met à jour un dépôt distant GitHub avec les modifications effectuées localement.

Code 25 Cas d'utilisation courants de **git push**

Exemples	Descriptions

git config --global push.default current	Pour utiliser le nom du dépôt local comme nom par défaut du dépôt distant vers lequel les modifications sont poussées
git push origin master	Repousse les modifications de la branche locale principale vers la branche distante principale
git push -d remote_repo_link nom_de_la_branche	Supprime une branche distante

3.1.5 git pull

git pull met à jour la branche courante dans le répertoire de travail avec les dernières modifications du serveur distant.

Cependant, cette commande peut entraîner des conflits de fusion.

Pour éviter cela :

- ne démarrer un **git pull** qu'avec un répertoire de travail propre
- sauvegarder temporairement les modifications du répertoire de travail en utilisant **git stash** ou **git worktree**
-

3.1.6 git fetch

git fetch télécharge les nouvelles données d'un dépôt distant, mais ne les intègre pas dans le répertoire de travail courant. Fetch est utile pour avoir une vue d'ensemble des changements survenus dans un dépôt GitHub.

Code 26 Récupérer un certain nombre de commits du dépôt distant GitHub

```
git fetch --depth=n
```

3.1.7 Fusionner

La fusion et le rebasage sont les principales méthodes d'intégration des modifications d'une branche à l'autre. Une troisième option est **git cherry-pick**, discutée à la page 36.

Code 27 Pour combiner les changements de la branche spécifiée avec la branche courante en utilisant **git merge**

```
git merge nom_de_la_branche
```

3.1.8 Rebasage

Le rebasage modifie l'emplacement d'un certain nombre de commits. Il peut également combiner et modifier les commits et leurs messages et peut également être utilisé pour réorganiser les commits. La version interactive de rebasing ouvre un éditeur interactif en ligne de commande et facilite ces fonctions.

```
git rebase -i
```

Contrairement à la fusion qui préserve l'historique de contrôle de version des livraisons, le rebasage le réécrit. Il convient donc d'utiliser le rebasage avec prudence.

3.1.9 Les conflits et leur résolution

Lors de l'intégration de modifications provenant de différentes branches à l'aide de la fusion ou du rebasage, des conflits peuvent survenir.

Par exemple, deux programmeurs peuvent modifier la même ligne de code de manière différente. Git ne sait pas quelle version est la bonne et ne peut que laisser des marqueurs dans les fichiers en conflit pour que les programmeurs les résolvent, comme le montre la figure 3.

Figure 3 Exemple d'une zone conflictuelle

```
<<<<<<< HEAD
        $("#new").click(function(){
=======
        $(el).find("#new").click(function (event) {
>>>>>>> Convert list to use presenter pattern.
          $("tr input[type=text]:first").focus();
        });
```

La commande **git diff** permet aux programmeurs de voir les conflits de fusion.

Code 28 Cas d'utilisation courants de **git diff**

Exemples	Descriptions
`git diff --base nom_du_fichier`	Affiche les conflits de fusion contre le fichier dans la branche courante
`git diff --ours nom_du_fichier`	Visualisation des conflits de fusion par rapport aux modifications apportées
`git diff --theirs nom_du_fichier`	Visualise les conflits de fusion par rapport à d'autres modifications
`git diff`	Permets au programmeur de comparer visuellement les changements entre différentes versions de fichiers ou de commits au sein d'un dépôt Git.

Il existe 3 méthodes pour résoudre les conflits de fusion.

Code 29 Méthodes de résolution des conflits de fusion

Méthodes	Descriptions
`git merge --abort`	Cette commande annule une fusion. Elle permet d'annuler et de recommencer
`git rebase --abort`	Cette commande annule un rebasement. Elle permet d'annuler et de recommencer

`1 : git add file1 file2`
`2 : git rebase -continue`
ou
`1 : git add file1 file2`
`2 : git merge --continue`
	Dans cette méthode, les conflits sont résolus et ajoutés, puis la fusion ou la refonte est reprise.

Unité 3.2
git cherry-pick



fr.learnapply.org/git/3.2

3.2.1 Aperçu

En utilisant **git cherry-pick, les** programmeurs peuvent sélectionner les commits à inclure lors de l'intégration des changements d'une branche à l'autre.

Il est également utile pour corriger les erreurs dans les commits précédents et pour appliquer les changements à plusieurs branches.

Cependant, elle peut introduire des conflits de fusion qui doivent être résolus manuellement et peut modifier l'historique des livraisons d'une branche.

3.2.2 4 règles pour l'utilisation de git cherry-pick

- Préférez la fusion ou le rebasage lorsque c'est possible. La fusion préserve l'historique complet des deux branches et le rebasage crée un historique propre et linéaire qui peut être plus facile à gérer et à réviser.
- Éviter de créer des commits dupliqués avec un contenu identique

- Utilisez **git cherry-pick** avec l'option **-x** car il fournit un filet de sécurité en interrompant automatiquement le processus en cas de conflit.
- Inclure un message de validation détaillé

3.2.3 Exemples

Ce sont des exemples de 4 cas d'utilisation impliquant le "cherry-picking".

ALGORITHME 4 Premier cas d'utilisation : Livrer une correction de bogue à l'utilisateur final le plus rapidement possible

Hypothèses :
La branche de production est la branche du programme déployée au public.
Le correctif a déjà été intégré dans la branche des fonctionnalités

1 : Identifier le commit qui contient la correction du bug en utilisant **git log**
2 : Passer à la branche de production en utilisant **git** checkout
3 : Utiliser git cherry-pick commit_hash pour appliquer la correction à la branche de production
4 : Résoudre les conflits éventuels en éditant la zone entre les marqueurs de flèche droite et de flèche gauche dans les fichiers.
5 : Pousser les changements vers le dépôt distant en utilisant **git push origin** HEAD
6 : Déployer les changements en utilisant le processus de développement

ALGORITHME 5 Deuxième cas d'utilisation : Annuler des modifications pour corriger des erreurs dans le commit précédent

Hypothèses :
Il existe une branche de production
Le correctif a déjà été intégré dans la branche des fonctionnalités

1 : Identifier le commit avec l'erreur à annuler en utilisant **git log**
2 : Créer un nouveau commit avec les changements inversés en utilisant **git cherry-pick commit_hash**
3 : Résoudre les conflits éventuels en modifiant la zone entre les marqueurs de flèche droite et de flèche gauche dans les fichiers.
4 : Pousser les changements vers le dépôt distant en utilisant **git push origin HEAD**

ALGORITHME 6 Troisième cas d'utilisation : Restaurer les commits perdus

1 : Identifier les commits perdus en utilisant **git reflog** (pas **git log**, car ils ne seront pas affichés)
2 : récupérer les commits perdus en utilisant **git cherry-pick commit_hash**
3 : Résoudre les conflits éventuels en modifiant la zone entre les marqueurs de flèche droite et de flèche gauche dans les fichiers
4 : Pousser les changements vers le dépôt distant en utilisant **git push origin HEAD**

ALGORITHME 7 Appliquer des modifications à plusieurs commits

Première méthode

1 : Identifier le ou les commit(s) contenant les changements à appliquer en utilisant **git log** et noter le ou les hash(s) du commit.
2 : Exécuter **git cherry-pick commit_hash** pour chaque commit

Deuxième méthode

1 : Identifier la plage de livraisons en utilisant le **journal git** pour trouver les hachages des premières et dernières livraisons d'une plage.
2 : Exécuter **git cherry-pick first_commit_hash..second_commit_hash**

Unité 3.3 Comprendre les Pull Requests de GitHub



fr.learnapply.org/git/3.3

3.3.1 Aperçu

Les Pull Requests, à ne pas confondre avec la commande **git pull**, sont initiées sur GitHub, et non en ligne de commande. Lorsqu'un programmeur

lance une Pull Request, il propose un ensemble de modifications que les autres membres de l'équipe doivent tester, réviser et commenter.

Si la décision d'appliquer les changements est prise, les responsables du dépôt les fusionneront. Au minimum, deux branches distinctes ou deux dépôts distincts sont nécessaires pour une demande d'extraction.

Une demande de retrait comporte les éléments suivants :

- Le titre et la description
- Les commits inclus dans la demande d'extraction
- Une section de discussion où les programmeurs peuvent fournir un retour d'information et des commentaires
- Une vue visuelle des modifications apportées au code

3.3.2 Mise en place d'une Pull Request

ALGORITHME 8 étapes pour créer une Pull Request

1 : Créer un dépôt à l'aide du site web GitHub
2 : Cloner le fork sur la machine locale en utilisant **git clone url**
3 : Créer une nouvelle branche en utilisant **git branch branch_name**
4 : Effectuer des changements localement et les ajouter en utilisant **git add**
5 : Valider les modifications en utilisant **git commit -m "commit_message"**
6 : Transférer les modifications en utilisant **git push --set-upstream origin** branch_name
7 : Mettez à jour le dépôt local en suivant ces 5 étapes

1. Trouver l'url du serveur distant en utilisant **git remote -v**
2. Spécifier le nouveau dépôt amont distant à synchroniser en utilisant **git remote add upstream url**
3. Synchroniser la fourche en utilisant **git fetch upstream.** Maintenant, les commits de la branche principale seront stockés dans une branche locale appelée upstream/main
4. Revenir à la branche principale locale du dépôt en utilisant **git checkout main**
5. Fusionner tous les changements effectués dans la branche principale du dépôt original à laquelle nous aurons accès par le biais de notre branche locale upstream/main avec notre branche principale locale en utilisant **git merge upstream/main.**

8 : Créer une nouvelle demande d'extraction en utilisant le site web de GitHub

3.3.3 Meilleures pratiques

- Les Pull Requests ne doivent avoir qu'un seul objectif et doivent être de petite taille afin d'améliorer la compréhensibilité et la facilité de révision.
- La branche forkée doit rester à jour par rapport à la branche principale afin de réduire les risques de conflits de fusion.
- Rédiger des descriptions détaillées
- Lien vers tous les problèmes pertinents. Si la demande d'extraction résout un problème ouvert, mentionnez-le
- Évitez de soumettre une demande trop tôt ; testez les changements et assurez-vous qu'ils fonctionnent comme prévu avant de demander à d'autres de les examiner.
- Soyez patient, car les responsables de la maintenance et les autres membres de l'équipe peuvent prendre du temps pour examiner les documents.
- Être ouvert au retour d'information et ne pas l'ignorer

Unité 3.4
Flux de travail



fr.learnapply.org/git/3.4

3.4.1 Aperçu

Un flux de collaboration standardise les étapes que les programmeurs d'une équipe suivent pour gérer et collaborer sur le code lorsqu'ils utilisent Git et GitHub. Il englobe des pratiques telles que le branchement, la fusion, l'examen du code, la description de la manière dont les modifications sont proposées et l'intégration des modifications approuvées dans la base de code principale.

L'utilisation d'un flux de travail permet de maintenir l'ordre et l'efficacité et d'améliorer la collaboration.

Les cinq flux de travail décrits dans ce sous-chapitre introduisent des conventions de dénomination pour les branches et l'objectif de chaque branche est prédéfini.

TableAU 2 Les différentes branches des flux de travail collaboratifs

Branches	Descriptions
main - principal	La version déployable de la base de code
feature - fonction	Pour les nouvelles fonctionnalités ou les corrections des erreurs
develop - développer	Servir de zone de transition pour le développement en cours
release - libération	Facilite la préparation des mises en production
hotfix - correctif	Pour résoudre des bogues ou des problèmes critiques
fork - fourchette	Une copie du dépôt principal créée par un programmeur

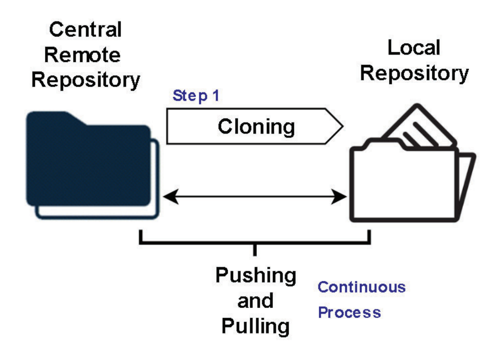
Ces flux de travail constituent un point de départ et les équipes peuvent créer leurs propres flux en fonction des besoins et des exigences d'un projet.

3.4.2 Centralized - Le flux de travail centralisé

Le flux de travail centralisé, également appelé flux de travail de base, commence par un dépôt central unique. Les programmeurs clonent le dépôt pour y apporter des modifications localement. Ensuite, ils poussent continuellement leurs modifications et tirent les modifications des autres programmeurs à partir de ce dépôt.

Des conflits potentiels peuvent survenir lorsque plusieurs programmeurs tentent d'apporter des modifications simultanément.

Figure 4 Vue d'ensemble du flux de travail centralisé



3.4.3 Le flux de travail de la Feature Branch

Ce flux de travail permet aux programmeurs de travailler de manière isolée par rapport à la base de code principale. Cela permet d'éviter les conflits avec la branche principale et de garantir sa stabilité. Dans ce flux de travail, une nouvelle branche est créée pour chaque nouvelle fonctionnalité ou correction de bogue.

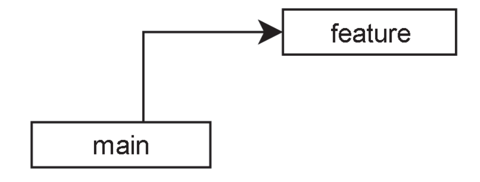
Les demandes d'extraction (pull requests) ou les fusions (merges) sont utilisées pour intégrer les fonctionnalités terminées ou les corrections de bogues dans la branche principale, ce qui crée un processus de développement continu (CD).

Une fois prête, la nouvelle fonctionnalité est poussée vers le dépôt distant GitHub. Elle est ensuite examinée et testée par d'autres membres. Si elle passe les tests, elle est fusionnée dans la branche principale. Une fois intégrée, la branche de la fonctionnalité peut être supprimée.

Ce flux de travail permet la simultanéité, ce qui permet à différents programmeurs de travailler sur différentes caractéristiques d'un projet en même temps.

Le modèle de développement ne comporte que deux branches : la branche principale et une ou plusieurs branches de fonctionnalités.

Figure 5 Branches du flux de travail Feature Branch



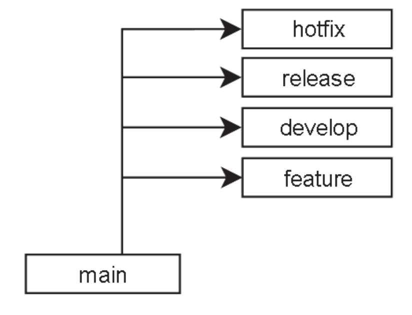
3.4.4 Le flux de travail GitFlow

Ce workflow est une extension du précédent workflow Feature Branch. Bien que l'objectif de la branche de fonctionnalité soit le même, GitFlow introduit plusieurs nouvelles branches et des conventions de nommage pour celles-ci.

La branche "hotfix" est destinée à corriger les bogues ou les problèmes critiques. La branche release facilite la préparation des versions de production. La branche develop sert de zone de transit pour le développement continu, car le code doit être maintenu et développé en permanence. La branche principale est destinée à la version déployable de la base de code.

Ce flux de travail convient aux projets dont les phases de développement et de mise en production sont clairement distinctes.

Figure 6 Branches du flux de travail GitFlow

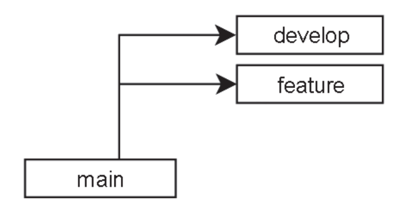


3.4.5 Le flux de travail GitHub Flow

Ce flux de travail est une version simplifiée du flux de travail GitFlow précédent. Les branches release et hotfix sont supprimées, mais les branches main, feature et develop sont conservées.

Ce flux de travail prend en compte la fonctionnalité Pull Requests de GitHub. Il convient si le programmeur utilise GitHub comme serveur distant pour les projets Git. Les programmeurs travaillent sur des branches de fonctionnalités et soumettent directement des demandes d'extraction à la branche principale.

Figure 7 Branches du flux de travail GitHub Flow



GitHub Flow encourage des versions plus petites et plus fréquentes et tente de réduire la complexité et la durée des branches de fonctionnalités, par conséquent, comme le flux de travail Feature Branch, il se concentre sur la livraison continue (CD).

ALGORITHME 9 Étapes du flux de travail GitHub Flow

1 : Créer une branche de fonctionnalité
2 : Travailler sur une fonctionnalité
3 : Pousser la branche de fonctionnalité et créer une demande de traction GitHub
4 : La Pull Request est examinée par les autres membres de l'équipe
5 : Si elle passe l'examen, elle sera intégrée
6 : Supprimer la branche de fonctionnalité

3.4.6 Le flux de travail Forking

Ce flux de travail crée une nouvelle version d'un dépôt qui est indépendante du programmeur. Cela se fait en forkant et en clonant le fork. Il permet un modèle de développement décentralisé qui autorise l'expérimentation avec les fourches avant de proposer des modifications au dépôt original. Cette approche décentralisée réduit les risques de conflits dans le dépôt principal.

Ce flux de travail ajoute une branche fork aux branches main, feature et develop de GitHub Flow.

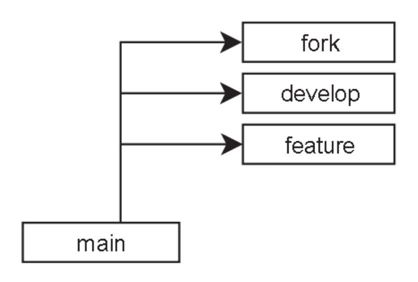
S'il est mis en œuvre, ce flux de travail nécessite 8 étapes :

ALGORITHME 10 étapes du flux de travail Forking

1 : Créer un dépôt principal (Fork)
2 : Cloner la fourchette sur la machine

3 : Créer une ou plusieurs branches de fonctionnalités pour ajouter des fonctionnalités et corriger des bogues
4 : Apporter des modifications
5 : Pousser la branche contenant les modifications vers le dépôt distant
6 : Créer une demande d'extraction
7 : La Pull Request est revue et testée par l'équipe
8 : Si le test est concluant, les modifications de la demande d'extraction sont fusionnées.

Figure 8 Branches de Forking



Unité 3.5
Comprendre les connexions SSH



fr.learnapply.org/git/3.5

3.5.1 Aperçu

Ce livre traite de l'utilisation d'URLs dans la ligne de commande pour accéder aux services de GitHub et aux dépôts distants. Secure Shell (SSH) est une méthode alternative. SSH utilise un cryptage fort pour protéger les données contre l'interception et est considéré comme plus sûr que les liens https.

Lors de la configuration, deux clés sont créées : une clé publique et une clé privée. La clé publique est partagée avec le serveur distant. Elle est utilisée pour vérifier l'identité du programmeur. La clé privée est gardée secrète.

3.5.2 Configuration d'une clé SSH

ALGORITHME 11 Étapes pour configurer une clé SSH pour Git et GitHub

1 : Ouvrez Git Bash et lancez **ssh-keygen**
2 : Ajouter la clé à Git Bash en exécutant ces deux commandes
    eval `ssh-agent -s ` ▷Assurez-vous d'utiliser ` et non '
    ssh-add ~/sshkey
3 : Voir la clé en utilisant **cat sshkey.pub** et la copier dans un endroit sûr
4 : Aller dans la section SSH et Clés GPG des paramètres du compte GitHub
5 : Définir une nouvelle clé SSH et sélectionner l'authentification comme type de clé

3.5.3 Exemples

Code 30 Exemples de SSH en ligne de commande

Exemples	Descriptions
`git clone git@github.com :[Nom d'utilisateur GitHub]/[Nom du repo].git`	Cloner un nouveau dépôt en utilisant SSH
`git remote set-url origin git@github.com :[nom d'utilisateur GitHub]/[nom du repo].git`	Changer un dépôt existant de https à SSH

Chapitre 4 Nettoyage, personnalisation et Correction des Erreurs

Contenu

Cette partie traite de la création d'abréviations personnalisées pour les commandes Git, de la recherche de fichiers et de commits perdus et de la rectification des erreurs. Elle fournit également des instructions sur le nettoyage des fichiers et des dépôts pour améliorer les performances et montre comment désactiver le suivi de Git pour des fichiers et des dossiers spécifiques.

Unité 4.1
Abréviations personnalisées pour les commandes Git



fr.learnapply.org/git/4.1

Un alias est une abréviation définie par le programmeur pour une commande Git plus longue. Ils permettent d'améliorer l'efficacité et de simplifier les flux de travail courants. Les alias sont définis dans le fichier de configuration de Git, .gitconfig.

Code 31 Raccourcis pour raccourcir le commit vers ci et pour déstocker les fichiers stagés en utilisant **git unstage**

Avant	Après
`git config --alias.ci "commit"`	**git ci -m "commit_message"**
`git config --alias.unstage "reset --"`	**git unstage**

Code 32 Pour lister tous les alias existants

```
git config --get-regexp '^alias\.
```

Unité 4.2
Corriger les erreurs



fr.learnapply.org/git/4.2

4.2.1 Retrouver les fichiers et les modifications perdus

Code 33 Pour lister tous les fichiers et commits perdus

```
git fsck --lost-found
```

Code 34 Contrôler les erreurs et nettoyer le dépôt en enlevant tous les objets inaccessibles et non fixés.

```
1. git fsck
2. git gc --prune
```

4.2.2 Recherche d'erreurs

ALGORITHME 12 Étape pour trouver le commit qui a introduit un bogue

1 : Démarrer une session de recherche binaire en utilisant **git bisect start**
2 : Donner à la session deux références : un bon commit avant le bug en utilisant **git bisect good commit_hash** et un mauvais commit après le bug en utilisant git **bisect bad commit_hash** ou **git bisect bad HEAD**

La session démarrera et Git trouvera l'endroit où le bogue a été introduit.

3 : Terminer la session en utilisant **git bisect reset**

4.2.3 Outils de correction

Les programmes de débogage WinMerge et Meld sont des alternatives à la commande `git bisect`.

Windows **winmerge.org**
Windows, macOS, Linux **meldmerge.org**

Unité 4.3
Nettoyer les fichiers et les dépôts



fr.learnapply.org/git/4.3

git clean supprime les fichiers et répertoires non suivis du répertoire de travail.

Un fichier ou un dossier non suivi est un fichier ou un dossier présent dans le répertoire de travail, mais qui ne fait pas partie du dépôt local, du dépôt distant ou de la zone de transit. Ses modifications et ses versions ne sont pas suivies par Git.

Code 35 Cas d'utilisation courants pour le nettoyage des fichiers et des répertoires

Exemples	Descriptions
`git clean -i`	Nettoyage interactif des fichiers et des dossiers
`git clean -f`	Supprime tous les fichiers non suivis
`git clean -fX`	Nettoie les fichiers ignorés du dossier actuel et de tous les sous-répertoires.
`git clean -fd`	Supprime tous les répertoires non suivis et les fichiers qu'ils contiennent.
`git clean -dn`	Prévisualise tous les répertoires qui seront nettoyés
1. `git fetch -p` 2. `git branch -vv`	Supprime le suivi entre les branches locales et les branches distantes supprimées. La deuxième commande affiche les branches qui ne sont plus suivies. Le mot "gone" indique qu'elles ne sont plus suivies.

Unité 4.4
Désactiver le suivi de Git



Un fichier ignoré ne le sera pas :

- Affiché dans les résultats de la ligne de commande de commandes telles que git **status** et **git diff**
- Ajouté à la zone de préparation ou validé en utilisant **git add** ou **git commit**

Il existe plusieurs méthodes pour supprimer un fichier du suivi de Git.

Code 36 Cas d'utilisation courants pour le nettoyage des fichiers et des répertoires

Exemples	Descriptions
`git config -global core.excludesfile .gitignore_file_path`	Ignore certains fichiers et dossiers de manière globale
`git rm --cached nom_du_fichier`	Ignore un fichier qui a déjà été déposé dans un dépôt Git
`git update-index --assume-unchanged nom_du_fichier`	Ignore les modifications ultérieures apportées à un fichier sans le supprimer du dépôt
`git update-index --skip-worktree nom_du_fichier`	Ignore les changements dans les fichiers suivis
`git status --ignored`	Recherche tous les fichiers du dossier courant ignorés par un fichier .gitignore global
`git check-ignore nom_du_fichier`	Vérifie si un certain fichier est ignoré

Figure 9 Exemple du contenu d'un fichier .gitingore

```
#this is gitignore file.
#list the pattern to instruct the git to ignore

/vendor                  # ignore the vendor directory
/node_modules            #ingnore node_modules directory
/public/storage          # ignore /public/storage directory
Homestead.yaml           #ignore Homestead.yaml file
Homestead.json           #ignore Homestead.json file
.env                     #ignore .env file
/nbproject/private/      #ignore /nbproject/private/ directory
/sql/*.sql               #ignore all .sql files in sql directory
```

Pour obtenir des modèles de fichiers .gitignore avec des exceptions généralement acceptées pour différents langages de programmation, systèmes d'exploitation et autres outils de programmation, consultez les sites web suivants

gitignore.io

github.com/github/gitingore

Chapitre 5
GitHub Actions et GitHub Pages

Contenu

Cette section explore deux services offerts par GitHub : Actions et Pages.

Unité 5.1
Comprendre les actions GitHub



fr.learnapply.org/git/5.1

5.1.1 Qu'est-ce que CI/CD ?

L'intégration continue (IC) consiste pour les programmeurs à intégrer en permanence dans le projet les modifications qu'ils apportent aux fichiers de code. Le développement continu (CD) prolonge l'intégration continue en automatisant le déploiement des modifications dans la production.

CI/CD est une pratique de développement de logiciels qui devrait être utilisée lors du développement de programmes à grande échelle plutôt qu'avec une petite base d'utilisateurs.

GitHub Actions est l'un des nombreux outils CI/CD disponibles. Pour une liste complète, visitez

github.com/cicdops/awesome-ciandcd

5.1.2 Qu'est-ce qu'une action GitHub ?

Une action GitHub est un morceau de code réutilisable qui s'exécute automatiquement en réponse à des événements dans le dépôt distant hébergé sur GitHub. Elles sont utilisées pour automatiser des tâches telles que la construction, le test, la validation et le déploiement de code. Ils réduisent le temps de développement et améliorent la qualité du code.

Les actions sont configurées dans des fichiers YAML, dont l'extension est .yaml ou .yml, et sont placées dans le sous-dossier .github/workflows d'un dépôt distant.

En plus des actions intégrées dans

github.com/actions

Les individus et les équipes peuvent publier leurs actions sur la place de marché publique pour que d'autres puissent les utiliser. Vous les trouverez à l'adresse suivante

github.com/marketplace?type=actions

5.1.3 Composants et fonctionnalités des actions GitHub

Les trois principaux composants d'une action GitHub sont les suivants :

- Flux de travail
- Emplois
- Les étapes

Les flux de travail définissent le processus global d'automatisation. Ils sont constitués d'une ou plusieurs tâches. Ils sont déclenchés par des événements tels que l'introduction de modifications, les demandes d'extraction ou les tâches planifiées.

Les tâches sont des tâches individuelles au sein d'un flux de travail. Ils exécutent des étapes pour mener à bien une tâche assignée. Ils peuvent être exécutés en parallèle ou de manière séquentielle en fonction de la configuration du flux de travail. GitHub met à la disposition des programmeurs des machines virtuelles (VM) pour l'exécution des tâches. Dans le contexte des actions GitHub, les machines virtuelles sont appelées "runners".

Les étapes sont les actions individuelles d'un travail. Une étape exécute une tâche. Ces actions sont soit intégrées, soit fournies par d'autres programmeurs, soit développées personnellement. La sortie de chaque étape peut être utilisée comme entrée pour les étapes suivantes du même travail.

Les actions GitHub offrent des possibilités de personnalisation supplémentaires qu'un programmeur peut utiliser.

TableAU 3 Fonctions fournies par GitHub Actions

Caractéristiques	Descriptions
Conditions	Contrôler l'exécution des tâches en fonction de déclencheurs. Par exemple, certaines modifications de fichiers

Matrices	Exécuter simultanément des tâches avec différentes configurations pour tester divers scénarios
Concurrence	Limite le nombre de tâches parallèles afin de gérer efficacement l'utilisation des ressources.
Environnements	Il est possible de définir différentes variables d'environnement pour différents travaux et étapes.
Conteneurs	Exécuter des tâches à l'intérieur de conteneurs tels que ceux fournis par Docker
Secrets	Stocker des informations sensibles telles que les mots de passe et les PAT
Reprises	Pour ré-exécuter certains travaux ou flux de travail si nécessaire

ALGORITHME 13 Étapes suivies pour exécuter une action GitHub

1 : Tout d'abord, un événement déclenche le flux de travail.
2 : Le flux de travail attribue des tâches individuelles sur la base de sa configuration.
3 : Exécution du travail sur la machine virtuelle choisie
4 : S'il est utilisé, le résultat de chaque étape est disponible pour les étapes suivantes du même travail.
5 : Le flux de travail se termine lorsque toutes les tâches ont été effectuées.

Unité 5.2
Comprendre les pages GitHub



fr.learnapply.org/git/5.2

GitHub Pages est un service d'hébergement de sites statiques. Les sites statiques ne génèrent pas de pages pour les programmeurs en temps réel et utilisent uniquement des fichiers HTML, CSS et JavaScript provenant d'un dépôt GitHub.

GitHub Pages exécute ces fichiers dans le processus de construction et fournit au programmeur une URL GitHub pour le site web.

Si un domaine personnel est configuré à la place, le programmeur devra configurer un enregistrement CNAME avec le registraire du domaine dans les paramètres DNS.

Lors de l'édition de l'enregistrement CNAME, le nom est le domaine acheté, et la valeur est l'URL fournie par GitHub Pages.

Il existe 3 types de pages GitHub :

- **Projet** : Les sites web sont connectés à un certain dépôt de projet hébergé sur GitHub.
- **Utilisateur** et **organisation** : Les sites web sont connectés à un certain compte GitHub.

Chapitre 6
Sujets Avancés et Supplémentaires

Contenu

Le dernier chapitre commence par une présentation de GitHub Desktop, puis explore les hooks, les submodules, les sous-arbres et les bundles. Il présente également la commande `git archive` qui permet de compresser les éléments Git dans un fichier ZIP. Le chapitre se termine par les meilleures pratiques d'utilisation de GitHub et de Git, ainsi que par une explication de la section README de GitHub.

Unité 6.1 GitHub Desktop



fr.learnapply.org/git/6.1

GitHub Desktop est un programme développé par GitHub. Plutôt que d'utiliser Git Bash ou la ligne de commande, il fournit une interface utilisateur graphique pour gérer l'interaction entre les dépôts locaux et distants hébergés sur GitHub.

Windows, macOS **desktop.github.com**

Il contribue aux fonctions suivantes :

- Création de nouvelles branches
- Fusionner les branches avec la branche principale
- Pousser, tirer et aller chercher changes
- Comparer et renommer les branches
- Changements dans les déchets et les réserves
- Combiner et fusionner les commits de la branche actuelle
- Rebasage de la branche actuelle
- Créer et prévisualiser les Pull Requests de GitHub

Le programme est présenté dans la vidéo de l'unité.

Unité 6.2 Crochets personnalisés



fr.learnapply.org/git/6.2

Similaires aux actions GitHub, les hooks sont des scripts qui s'exécutent automatiquement si un certain événement ou une certaine action les déclenche. Les fichiers de hooks ont des noms prédéterminés qui ne peuvent pas être modifiés. Ils ne prennent pas d'extension de fichier et doivent être placés dans le sous-dossier .git/hooks d'un dépôt Git.

Les scripts sont soit côté serveur, soit côté client.

6.2.1 Scripts côté serveur

Les scripts côté serveur sont déclenchés par des opérations réseau telles que la réception de "push commits".

TableAU 4 Exemples de hooks Git côté serveur et leur utilité

Scripts	Descriptions
update	Déclenché après une mise à jour du dépôt
post-commit	Exécuté après un commit. Il est généralement utilisé pour notifier les programmeurs de Git.
post-receive	Pour notifier les programmeurs Git. Il est exécuté après la réception d'un push
pre-commit	Déclenchée avant qu'une livraison ne soit effectuée, elle sert généralement à exécuter des tests automatisés afin de s'assurer que la livraison ne brisera pas le projet.
pre-commit-msg	Déclenché avant l'ouverture de l'éditeur de messages de validation
pre-base	Exécuté avant le début d'un rebasement. Généralement utilisé pour s'assurer que le rebasement est approprié
pre-receive	Exécuté avant la réception d'un "push".

6.2.2 Scripts côté client

Ces scripts sont déclenchés par des opérations normales telles que la validation et la fusion.

Un exemple est le pré-push

Il est déclenché avant qu'une poussée ne soit effectuée vers un autre dépôt. Il est généralement utilisé pour empêcher le push d'atteindre le dépôt distant.

Pour plus d'informations sur les crochets Git, consultez le site suivant

git-scm.com/docs/githooks

Unité 6.3
Sous-modules et sous-arbres



fr.learnapply.org/git/6.3

Les sous-modules et les sous-arbres sont des méthodes permettant d'intégrer un dépôt Git à l'intérieur d'un autre dépôt Git. Ils sont utilisés pour gérer les dépendances de code d'un projet.

TableAU 5 Différences entre les sous-modules et les sous-arbres

Sous-modules	Sous-arbres
Un sous-module sert de lien vers des dépôts externes.	Les sous-arbres intègrent l'ensemble du contenu d'un autre dépôt, y compris son historique, dans le dépôt principal.
Un sous-module permet de réduire la taille du dépôt principal, car il ne contient que le lien vers le code externe.	Une sous-arborescence augmente considérablement la taille du dépôt principal, car elle ne comprend que tous les fichiers et l'historique du sous-projet.

La commande **git submodule permet** de gérer les sous-modules.

Code 37 Cas d'utilisation courants du sous-module **git submodule**

Exemples	Descriptions
`git submodule update --recursive --remote`	Extrait tous les sous-modules de leurs dépôts distants respectifs
`git submodule add url local_folder_path`	Ajoute un nouveau sous-module au dépôt local
1. `git submodule deinit -f --name submodule_name` 2. `rm -rf .git/modules/sous-module_name`	Supprime un sous-module du dépôt

`3. git rm -f nom_du_sous-module`	
`git submodule update --init --recursive`	Clone les sous-modules manquants et les commits de checkout

Unité 6.4
Liasses



fr.learnapply.org/git/6.4

Les bundles aident les programmeurs à transférer des dépôts vers des machines qui n'ont pas de connexion internet. Ils permettent d'empaqueter tous les objets Git et les références d'un dépôt dans un bundle qui peut être transféré vers une autre machine via une clé USB par exemple. Leur extension de fichier est .bundle

Code 38 Crée un fichier de liasse qui peut être transféré sur une autre machine.

```
git bundle create nom_du_bundle.bundle
```

Unité 6.5
Créer des archives



fr.learnapply.org/git/6.5

Une archive contient des fichiers et des dossiers d'un dépôt Git, d'une branche ou d'une balise, regroupés dans un fichier .zip. Il s'agit d'un processus similaire à la compression à l'aide de WinRAR ou de l'outil ZIP intégré de Windows.

Une archive permet de partager l'intégralité du code base à un moment donné sans avoir besoin d'un clone Git complet.

Code 39 Cas d'utilisation courants de l'archive **git**

Exemples	Descriptions
`git archive --format zip HEAD > archive-HEAD.zip`	Crée une archive ZIP de la branche courante
`git archive --output=archive-archive_name -prefix=src-directory=name tag_name`	Crée une archive ZIP d'une certaine balise. Le nom de l'archive et de la balise doivent être définis à l'adresse .
`git archive --output=archive-nom_archive -prefix=répertoire_src=nom nom_branche_locale`	Crée une archive ZIP d'une branche locale. Le nom de l'archive et de la branche locale doivent être définis à l'adresse .
`git archive --output=archive-archive_name -prefix=src-directory=name remote_branch_name`	Crée une archive ZIP d'une branche distante. Le nom de l'archive et de la branche distante doivent être définis à l'adresse .

Unité 6.6
Les bonnes pratiques et la section README de GitHub



fr.learnapply.org/git/6.6

6.6.1 Gestion des branches

Pour gérer efficacement les branches, il faut les concentrer sur l'accomplissement d'une certaine tâche, comme l'ajout d'une nouvelle fonctionnalité ou la correction d'un bogue.

Les meilleures pratiques pour gérer efficacement les succursales sont les suivantes :

- Mettre régulièrement à jour les branches
- Donner des noms significatifs aux branches
- Veiller à ce que les branches servent un objectif précis, tel que l'ajout d'une nouvelle fonctionnalité ou la correction d'un bogue.

- Supprimer les branches fusionnées
- Utilisez les règles de protection de GitHub, qui garantissent certains flux de travail pour une ou plusieurs branches, par exemple en exigeant et en approuvant une révision des modifications du code.

6.6.2 Organiser les dépôts

Les meilleures pratiques pour l'organisation des dépôts sont les suivantes :

- Utiliser une convention de dénomination claire et descriptive
- Regrouper les fichiers apparentés dans des dossiers pertinents
- Créer un fichier README détaillant l'objectif d'un dépôt
- Utiliser des balises pour des publications distinctes
- Squasher les commits avant de fusionner les branches
- Rédiger de bons messages d'engagement
- Utiliser des outils de suivi des problèmes pour gérer les buissons et les demandes de fonctionnalités
- Utiliser le système de stockage de fichiers volumineux (LFS) de Git pour les fichiers volumineux

6.6.3 La section README

Le fichier README.md est placé dans le dossier principal d'un dépôt distant. Il contient du code markdown qui est compilé pour générer une section organisée à la fin de la page principale d'un dépôt GitHub.

Les responsables d'un projet l'utilisent généralement pour mentionner des informations importantes sur le projet, telles que des instructions d'installation, des explications et d'autres informations diverses.

Voici la section README de PyTorch, un cadre d'apprentissage automatique open-source développé par Facebook.

github.com/pytorch/pytorch#readme

Annexe

fr.learnapply.org/cours/annexe

Contenu

L'annexe fournit des ressources de programmation de recherche supplémentaires. Une explication des codes de correction d'erreur de Reed-Solomon qui sont utilisés en cryptographie est donnée. Ces codes facilitent le recouvrement des codes QR. Les ressources utilisées pour créer ce livre et son cours vidéo sont listées, ainsi que plusieurs opérations de ligne de commande sur Linux.

Le livre finit par les autres cours, livres et séries d'Apprends et Applique.

Ressources additionnelles

C'est en forgeant qu'on devient forgeron. Appliquez le contenu de ce livre à un projet personnel et essayez d'utiliser GitHub et Git dans tous vos futurs développements logiciels, quel que soit le langage de programmation.

Pour plus de détails sur les options disponibles pour les commandes Git et les fonctionnalités de GitHub, consultez les documentations officielles:

git-scm.com/docs

docs.github.com

Des sites web de documentation supplémentaires:

overapi.com

devdocs.io

Des forums pour poser des questions sur la programmation et demander de l'aide pour corriger des erreurs:

stackoverflow.com

support.github.com

D’autres ressources:

github.com/dictcp/awesome-git

github.com/compscilauren/awesome-git-hooks

github.com/stevemao/awesome-git-addons

github.com/git-tips/tips

github.com/arslanbilal/git-cheat-sheet

github.com/matchai/awesome-pinned-gists

github.com/tiimgreen/github-cheat-sheet

github.com/stefanbuck/awesome-browser-extensions-for-github

github.com/cicdops/awesome-ciandcd

Généralement, si vous souhaitez en savoir plus sur un outil ou un concept de programmation, recherchez **`GitHub Awesome`** suivi de son nom.

Par exemple, vous pouvez trouver d'autres outils, frameworks, API et bibliothèques pour GitHub et Git sur le dépôt GitHub Awesome:

github.com/dictcp/awesome-git

Des plateformes pour les projets open-source et les outils de développement:

sourceforge.net

github.com

Des remises sont régulièrement proposées pour les produits commerciaux.

En utilisant l'adresse électronique de leur école, les étudiants peuvent demander une réduction de prix ou un accès gratuit à divers outils de programmation et avantages par le biais de programmes tels que le GitHub Developer Pack, ou en contactant directement les développeurs.

Table 6 Ressources additionnelles

Ressource	Description
Hugging Face **huggingface.co**	Accueille des projets d'Intelligence Artificielle (IA)
Google Gemini **gemini.google.com**	Un Grand Modèle Linguistique LLM pour la génération de contenu
Microsoft Copilot **copilot.microsoft.com**	Une alternative au LLM de Microsoft
ChatGPT **chat.openai.com**	Un LLM alternatif par OpenAI
massCode **masscode.io**	Un gestionnaire d'extraits de code pour les développeurs
Google Books Ngram Viewer **books.google.com/ngrams**	Suivi de la fréquence des mots et des phrases dans des millions de livres
Google Trends **trends.google.com**	Révèle la popularité des mots clés recherchés sur Google

Obsidian **obsidian.md**	Stocke les pensées et les idées
LogSeq **logseq.com**	Un alternatif à Obsidian
Google Scholar **scholar.google.com**	Un moteur de recherche pour les documents de recherche
ArXiv **arxiv.org**	Une archive de millions d'articles de recherche et de prépublications en libre accès gérée par l'Université Cornell
Articles ArXiv-Sanity **arxiv-sanity-lite.com**	Fournit une interface alternative pour ArXiv
CatalyzeX **catalyzex.com**	Fournit des implémentations de code pour la plupart des articles ArXiv
PapersWithCode **paperswithcode.com**	Un portail pour les articles sur l'IA, leur code et les ensembles de données
IEEE Xplore **ieeexplore.ieee.org/Xplore/home.jsp** **ACM Bibliothèque Numérique I Digital Library** **dl.acm.org** **Science Direct** **sciencedirect.com** **JSTOR** **jstor.org** **Scopus** **scopus.com** **Web of Science** **webofscience.com/wos** **DOAJ**	Une liste de bases de données contenant des documents de recherche sur divers domaines et sujets

doaj.org	
Kaggle **kaggle.com**	Une plateforme pour l'IA et la science des données
Google Public Data **google.com/publicdata**	Visualisation d'ensembles de données accessibles au public
TensorBoard par TensorFlow **tensorflow.org/tensorboard**	Visualisation et outils nécessaires aux expériences d'IA
Connected Papers **connectedpapers.com**	Fournit un visuel permettant de voir les liens entre les papiers de recherche
Aim **aimstack.io**	Un traqueur d'expériences open-source basé sur l'IA
Dimensions **app.dimensions.ai/auth/base/landing**	Visualisation et navigation dans de vastes réseaux d'articles pour découvrir les liens entre eux
Phind **phind.com**	Moteur de recherche basé sur l'IA et programmeur de couples
Perplexity.ai **perplexity.ai**	Un moteur de recherche alternatif basé sur l'IA
Elicit **elicit.com**	Analyser les documents de recherche et les résumer
Contrôle de la version des données | Data Version Control **dvc.org**	Un système de contrôle de version open-source pour les projets de science des données et d'IA
InterviewBit **interviewbit.com**	Fournit des questions d'entretien gratuites et leurs solutions
AlgoExpert **algoexpert.io**	Une ressource payante. Elle fournit des questions d'entretien de codage et leurs solutions
MLFlow Tracking **mlflow.org/docs/latest/tracking.html**	Fournit un API pour voir les détails et visualiser les résultats des expériments d'IA
ClearML **clear.ml**	Une plateforme pour gérer l'ensemble du cycle de vie de l'IA

DAGsHub **dagshub.com**	Une plateforme de collaboration open-source pour la science des données
Krater.ai **krater.ai**	Une application d'IA pour les créateurs de contenu et les rédacteurs
Chatter avec des LLMs **chat.lmsys.org**	Un projet qui évalue la précision des réponses des LLM aux demandes de renseignements

Des livres comme lectures supplémentaires:

Git & GitHub les premiers pas de David Hockley **amazon.fr/dp/B0B4G37CG1**

Maîtriser Git et GitHub : Du Débutant à l'Expert de Busy Engi **amazon.fr/dp/B0CK3M4TFC**

Pour plus de détails et d'autres ressources, effectuez des recherches sur Google, Bing, YouTube, etc.

Les codes de correction d'erreur Reed-Solomon (codes QR)

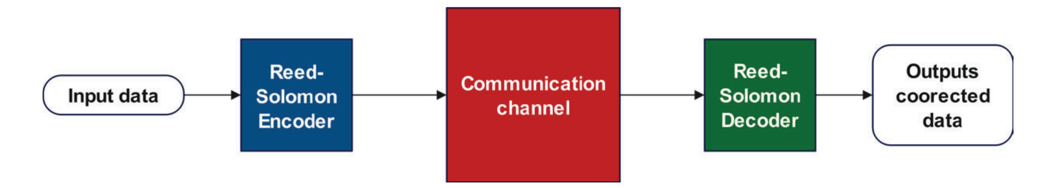
La correction d'erreurs fait référence à la détection et à la correction automatiques des erreurs dans les données. Les erreurs se produisent lors de la transmission incorrecte des données ou à la suite d'une corruption des données.

Pour générer des codes QR, les auteurs ont utilisé l'extension QR4Office pour Microsoft Word et le site web

qrbatch.com

Ils utilisent les codes de correction d'erreur Reed-Solomon qui ajoutent des bits supplémentaires au code QR afin qu'il puisse être corrigé s'il est corrompu.

Figure 10: Un système Reed-Solomon typique



Dans un système Reed-Solomon, le codeur prend des données numériques en entrée et y ajoute des bits redondants (sous forme de symboles) pour faciliter le processus de récupération en cas de corruption.

Des erreurs se produisent dans les données en raison du bruit, des interférences et des rayures. Dans le cas des codes QR, ils peuvent être corrompus par l'usure, les taches, les marques, l'impression en basse résolution ou la décoloration.

Le décodeur traite chaque bloc de données et tente de corriger les erreurs et de récupérer les données d'origine. La quantité de données pouvant être corrigées dépend des caractéristiques du code Reed-Solomon.

QR4Office permet aux utilisateurs de sélectionner l'une des corrections d'erreur suivantes:

- Faible – 7%
- Moyen – 15%
- Quartile – 25%
- Haut – 30%

Ces taux se réfèrent au niveau de correction des erreurs (ECL) et représentent le pourcentage maximum de données qui peuvent être corrigées en cas d'erreurs.

Au cours du processus d'encodage, des algorithmes et des concepts mathématiques, tels que la théorie de Galois, déterminent la longueur de ces symboles et l'endroit où ils sont ajoutés.

La théorie de Galois est à la base de la cryptographie en général et de la correction d'erreurs et de Reed-Solomon en particulier.

Pour plus de détail, nous vous recommandons de lire cette introduction par Lamrani Imane:

memoirepfe.fst-usmba.ac.ma/download/501/pdf/501.pdf

Références

Atlassian. (n.d.). Git Tutorials and Training | Atlassian Git, Tutorial. Atlassian de
atlassian.com/git/tutorials

awesome-cheatsheets/tools/git.sh chez master LeCoupa/awesome-cheatsheets. (2023, 21 Décembre). GitHub de
github.com/LeCoupa/awesome-cheatsheets/blob/master/tools/git.sh

Don, E. (2023). Git prodigy : maîtriser le contrôle de version avec Git et Github.

Livre Git gratuit. (2023, 21 Décembre). Books.goalkicker.com de
books.goalkicker.com/GitBook

Git Bash de
git-scm.com

L'antisèche Git. (2023, 21 Décembre) de
cheat-sheets.org/saved-copy/github-git-cheat-sheet-20220806.pdf

Aide-mémoire Git. (2023, 21 Décembre). GitHub de
cheat-sheets.org/saved-copy/git-cheat-sheet-v2.pdf

Git Cheat Sheet Créer. (2023, 21 Décembre) de
cheat-sheets.org/saved-copy/git-cheat-sheet.pdf

Référence rapide Git. (2023, 21 Décembre) de
cheat-sheets.org/saved-copy/Git_Quick_Reference.2011-09-04.pdf

Tutoriel Git. (2023, 2 Mai). GeeksforGeeks de
geeksforgeeks.org/git-tutorial

Tutoriel Git. (2023, 23 Décembre) de
w3schools.com/git

Google. (2024). Gemini [Grand Modèle Linguistique] de
gemini.google.com

Homebrew de
brew.sh

Lodha, K. (2023, 31 Janvier). Git Describe. Scaler Topics de
scaler.com/topics/git/git-describe

Microsoft. (2024). Copilot [Grand Modèle Linguistique] de
copilot.microsoft.com

OpenAI. (2023). ChatGPT [Grand Modèle Linguistique] de
chat.openai.com

codes de reed-solomon. (2019). Cmu.edu de
cs.cmu.edu/~guyb/realworld/reedsolomon/reed_solomon_codes.html

ET DE L'INSTANTANÉ. (2023, 21 Décembre) de
cheat-sheets.org/saved-copy/git-cheat-sheet-education.20211117.pdf

Les 20 premières commandes Git avec des exemples - DZone. (2023, 21 décembre). Dzone.com de
dzone.com/articles/top-20-git-commands-with-examples

Visual Studio Code de
code.visualstudio.com

Liste d’opérations de ligne de commande sur Linux

Voici une liste complète des commandes les plus couramment utilisées dans la ligne de commande de Linux. La plupart de ces opérations ont des arguments pour différentes tâches. Recherchez les commandes qui vous intéressent pour plus de détails.

sudo
▷Pour donner à l'utilisateur actuel l'accès à la racine (l'équivalent de "Exécuter en tant qu'administrateur" dans Windows). Cette option est souvent utilisée pour les commandes qui nécessitent un accès au système de fichiers

apt
▷Pour gérer les paquets sur les systèmes Linux basés sur Debian

snap
▷Pour gérer les paquets snap

apropos
▷Pour trouver toutes les commandes en ligne de commande contenant un certain mot clé

man
▷Pour afficher la page de manuel d'une certaine ligne de commande

whatis
▷Fournir une brève description d'une commande en ligne

cd
▷Pour naviguer entre les dossiers

ls
▷Pour dresser la liste de tous les fichiers et dossiers situés dans le dossier actuel

mkdir
▷Pour créer un nouveau dossier à l'intérieur du dossier actuel

touch
▷Pour créer un nouveau fichier dans le dossier actuel

pwd
▷Pour afficher le nom du dossier en cours

tail
▷Pour afficher les dernières lignes d'un fichier

head
▷Pour afficher les premières lignes d'un fichier

diff
▷Comparer le contenu de 2 fichiers ligne par ligne et afficher les différences entre eux

uniq
▷Pour filtrer les lignes en double dans un fichier

cmp
▷Tester si 2 fichiers sont identiques ou non

comm
▷Pour comparer 2 fichiers triés et afficher les lignes communes aux deux fichiers

chmod
▷Pour modifier les autorisations d'un fichier ou d'un dossier

sudo find
▷Pour rechercher des fichiers dans le système

lsattr
▷Pour lister les propriétés d'un fichier

chattr
▷Pour modifier les propriétés d'un fichier

open
▷Pour ouvrir un fichier situé dans le dossier actuel

cat
▷Pour imprimer le contenu d'un petit fichier dans la ligne de commande

less

▷Pour imprimer le contenu d'un fichier volumineux dans le dossier actuel

cp

▷Pour copier et coller des fichiers et des dossiers d'un dossier à l'autre

rm

▷Pour supprimer un fichier ou un dossier dans le dossier actuel

nano

▷Pour créer et ouvrir un nouveau fichier dans le dossier actuel

grep

▷Pour trouver du texte dans un fichier situé dans le dossier actuel

echo

▷Écrire du texte dans un fichier

mv

▷Déplacer ou renommer des fichiers et des dossiers

history

▷Pour afficher toutes les commandes exécutées précédemment dans la session actuelle de la ligne de commande.

ping

▷Pour vérifier si un hôte du réseau est joignable ou non

ssh
▷Pour se connecter à un système distant

w
▷Pour afficher des informations sur les utilisateurs actuellement connectés

uptime
▷Pour savoir depuis combien de temps le système fonctionne

free
▷Pour afficher des informations sur l'utilisation de la mémoire du système

du
▷Pour afficher l'utilisation de l'espace disque des fichiers et des dossiers

df
▷Pour afficher des informations sur l'utilisation de l'espace disque du système

mkfs.ext4
▷Pour créer un nouveau système de fichiers ext4 sur une partition de disque ou un disque séparé

1. **sudo mount**
2. **sudo fsck**
3. **sudo unmount**

▷Pour récupérer et réparer un système de fichiers. Le système de fichiers doit d'abord être monté. À la fin, il doit être démonté

zip
▷Pour créer un fichier zip

gzip
▷Pour créer un fichier gzip

unzip
▷Pour extraire un fichier zip dans le dossier courant

gunzip
▷Pour extraire un fichier gzip dans le dossier courant

watch
▷Exécuter de manière répétée un certain script ou une certaine commande à un intervalle de temps fixe

sleep
▷Introduire un délai avant l'exécution d'un script ou d'une commande

dd
▷Pour copier des fichiers d'un dossier à un autre. Cette commande peut également être utilisée pour convertir différents types de fichiers

rsync
▷Transférer des dossiers entre deux emplacements

sync
▷Pour s'assurer que toutes les données en attente sont écrites dans la mémoire

chown
▷Pour changer le propriétaire ou le groupe du fichier

chgrp
▷Pour modifier la propriété du groupe d'un fichier ou d'un dossier

env
▷Pour lister les variables d'environnement

alias
▷Créer des raccourcis pour les commandes

whereis
▷Pour localiser les fichiers exécutables et binaires d'une commande

groupadd
▷Pour créer un groupe d'utilisateurs

members
▷Pour dresser la liste des membres d'un groupe

groupmod
▷Modifier les propriétés d'un groupe

groupdel
▷Pour supprimer un groupe d'utilisateurs

usermod
▷Modifier les propriétés d'un utilisateur

dump
▷Créer des sauvegardes de fichiers ou de systèmes de fichiers

restore
▷Pour restaurer les sauvegardes

source
▷Pour exécuter un script shell qui se trouve dans le dossier courant

jobs
▷Pour lister toutes les tâches d'arrière-plan de la session en cours

ps
▷Pour lister tous les processus en cours

top
▷Pour avoir une vue en temps réel de tous les processus

htop
▷Pour obtenir une vue interactive de tous les processus

kill
▷Tuer le processus d'un programme

kill -9
▷Pour forcer l'arrêt d'un processus de programme

killall
▷Tuer tous les processus d'un programme

clear
▷Pour effacer de la ligne de commande tout ce qui y est affiché

exit
▷Pour quitter la ligne de commande

Les autres cours et livres d’Apprends et Applique

Le cours et le livre *GitHub et Git: Un Guide* font partie de la série **Programmation**. Cette série couvre l'Intelligence Artificielle, Python, JavaScript, C++, les bases de données, le développement d'applications et d'autres sujets.

La plateforme Apprends et Applique comprend une variété d'autres séries tels que Finance et Comptabilité, Mathématiques et Statistiques, Sciences, Humanités et Langues.



fr.LearnApply.org

Le contenu est disponible en différentes langues; les cours, les livres et les livres audio portant le symbole **β** ont été traduits et leur audio a été généré à l'aide d'outils basés sur l'Intelligence Artificielle.

Pour améliorer ces traductions et ces voix off ou pour aider à la création de nouvelles langues:

fr.learnapply.org/traduire

Pour demander de l'aide avec un projet personnel ou pour rester en contact et recevoir des mises à jour sur les nouveaux cours et livres d’Apprends et Applique, rejoignez les communautés de Reddit et Discord et abonnez-vous au bulletin électronique:

fr.learnapply.org/social

Pour faire un don et nous aider à couvrir les frais de publicité, de traduction, d'impression et les autres couts:

fr.dridi.org/donner

Apprends et Applique est un projet éducatif produit par La Fondation Dridi. Les projets de la fondation sont dédiés au plus grand bien économique et social de l'humanité

fr.dridi.org

GitHub et Git: Un Guide

Ce livre explique la plupart des sujets liés à GitHub et au système de contrôle de version Git; il décrit également de nombreuses commandes Git et fournit des détails sur les cas d'utilisation et les options les plus populaires.

Une expérience précédente avec GitHub ou Git n'est pas nécessaire.

Le livre explique également les avantages de l'utilisation de Git pour gérer les fichiers de code, tels que la gestion des branches et l'historique.

Quelques commandes expliquées

- git init
- git status
- git add
- git commit
- git branch
- git mv
- git rm
- git log
- git tag
- git describe
- git diff
- git blame
- git show
- git checkout
- git switch
- git restore
- git remote
- git clone
- git push
- git pull

Git peut être utilisé avec des répertoires distants pour faciliter le travail d'équipe et la collaboration. GitHub rend cela possible et offre des services supplémentaires tels que Actions et Pages. Ce livre couvre GitHub à partir du Chapitre 3.

Apprends et Applique est une plateforme éducative multilingue concise, accessible et gratuite développée par La Fondation Dridi. L'impact et les productions de la fondation sont le fruit d'un effort collectif de M. Ridha Dridi, Mme Fathia Jelassi Ep Dridi, Aziz Dridi et Salim Dridi. Les familles Dridi et Jelassi viens de la Tunisie.

Les projets de la fondation sont dédiés au plus grand bien économique et social de l'humanité

fr.Dridi.org

Série de Programmation



Apprends et Applique
fr.LearnApply.org

